Rapport de projet

Isola

Réalisé par
Edouard BREUILLE
Célia ROUQUAIROL
Thibaut RUIZ
Jérémy ZAMIA

Sous la direction de
Vincent BOUDET

Année universitaire 2014 - 2015

Sommaire

Table des matières

1 – Cahier des charges 2
1.1 – Présentation du sujet 2
1.2 – Analyse des besoins fonctionnels 2
1.3 Analyse des besoins non fonctionnels 4
2 – Rapport technique 5
2.1 – Conception 5
2.2 – Résultat 10
2.3 – Perspectives de développement 10
4 – Rapport d'activité 12
4.1 – Cycle de développement 12
4.2 – Planification 13

Table des figures

Figure 1 Première version du diagramme des classes 3
Figure 2 Algorithme de pondération des cases..... 6
Figure 3 Exemple de parcours "bord après bord"..... 6
Figure 4 Correspondance entre les cases logiques et graphiques..... 7
Figure 5 Choix de la case de départ de l'IA 8
Figure 6 Initialisation des éléments graphiques 9
Figure 7 Action des boutons du menu..... 9
Figure 8 Diagramme de séquence du tour basique d'un joueur 10
Figure 9 Cycle de développement 12
Figure 10 Planning prévisionnel et réel du projet..... 13

Introduction

Isola est un jeu de plateau opposant deux joueurs (réels ou virtuels). Chacun joue son tour, qui consiste en deux étapes :

- Déplacer son pion sur l'une des huit cases blanches à proximité à condition qu'elles ne soient pas occupées par l'autre joueur ;
- Bloquer l'accès à une case blanche non occupée en y plaçant un carré noir.

Le premier joueur ne pouvant plus déplacer son pion est déclaré perdant.

Notre objectif est donc d'implémenter cette description en incluant une partie graphique.

Premièrement, le cahier des charges présentera de manière précise toutes les conditions à remplir, qu'elles aient été fixées de base ou bien ajoutées par nous-même.

Ensuite le rapport technique abordera les aspects plus concrets du projet, de sa conception sur papier aux développements possibles pour le futur.

Le manuel d'utilisation quant à lui aura pour rôle de détailler l'installation du jeu ainsi que comment y jouer.

Enfin, le rapport d'activité relatera d'une partie plus personnelle puisqu'il détaillera notre organisation et notre ressenti.

1 – Cahier des charges

1.1 – Présentation du sujet

Dans ce projet, il faut partir de zéro. Le sujet permet une grande liberté au niveau des choix fonctionnels. Bien qu'une étude de faisabilité ait été faite, nous avons déterminé nous-même par la suite les outils dont nous nous sommes servis tout au long de ces derniers mois. Ils ont bien sûr été validés par notre tuteur, Vincent BOUDET.

Nous avons fait le pari de transformer ce simple projet d'étude en expérience personnelle. Plus que réaliser une maquette d'un jeu, nous avons pris le parti de travailler avec sérieux et préparer notre projet à des évolutions futures. Nous continuerons de le faire évoluer au fur et à mesure des techniques apprises.

1.2 – Analyse des besoins fonctionnels

Afin de bien comprendre le fonctionnement du jeu, nous avons au départ joué plusieurs parties entre nous. Cela nous a permis d'avoir quelques pistes de réflexion et d'avoir une première approche du comportement de l'IA.

Nous avons ensuite dressé un premier diagramme de classes qui correspondait à notre idée du jeu.

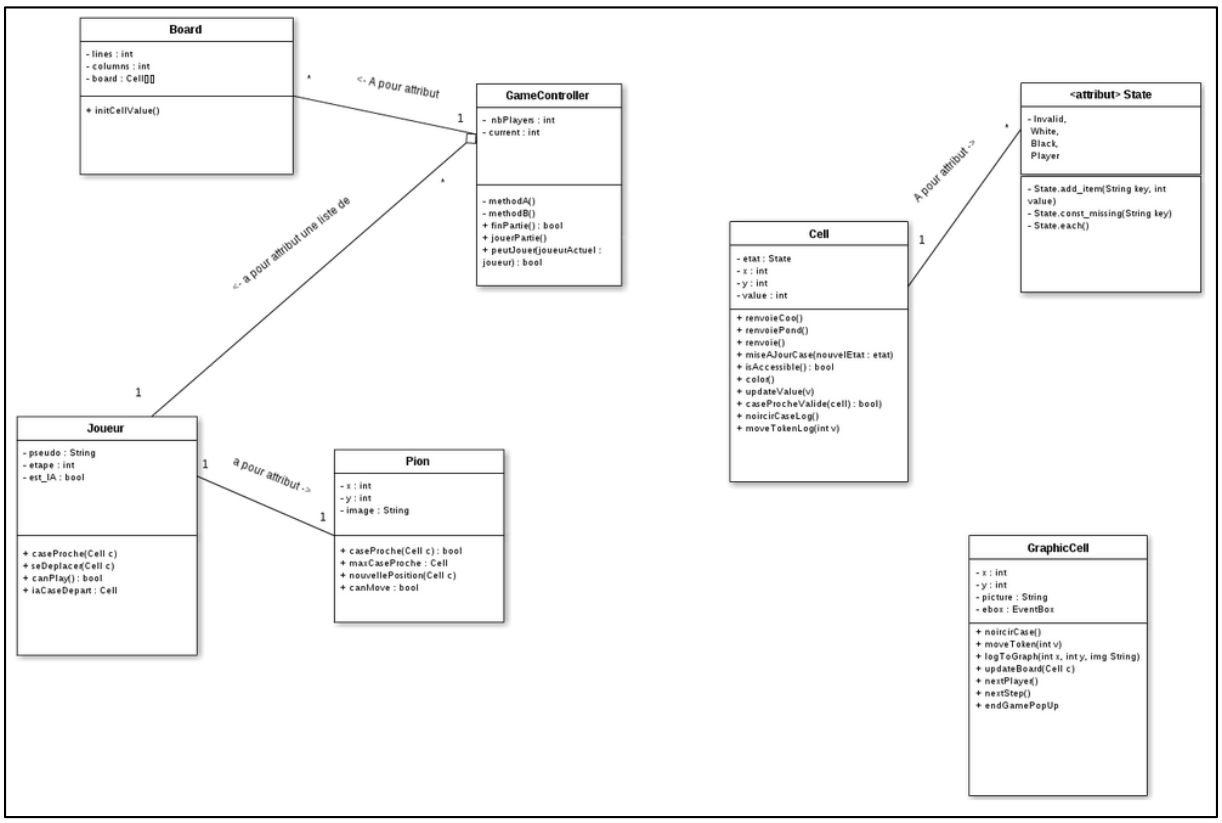


Figure 1 Première version du diagramme des classes

Ayant décidé de progresser par étape, notre premier objectif fut de réaliser une version « joueur contre joueur ».

- La classe Partie : elle regroupe la liste des joueurs ainsi que le plateau. Elle est chargée de gérer le bon fonctionnement du jeu.
- La classe Joueur : on regroupe ici tous les traits communs à des joueurs virtuels ou réels. Deux autres classes découleront de celle-ci par la suite : Joueur humain et Joueur IA, qui auront en plus les éléments caractéristiques à leur type de joueur.
- Chaque joueur possède un Pion : afin d'alléger le contenu des classes des joueurs ainsi que de par notre vision, nous en avons fait une classe à part entière.
- Il y a également un Plateau : là où se déplacent les joueurs, ses dimensions étant nécessaires pour les différents parcours à effectuer.
- Chaque case du plateau est une instance de la classe Cell : de même que le pion, il a été décidé d'attribuer aux cases des éléments qui leur étaient propre, comme leurs coordonnées ou leur état (blanc, noir, occupée).

1.3 Analyse des besoins non fonctionnels

1.3.1 – Spécifications techniques

Comme dit précédemment, nous avons été très libres pour ce qui est des choix techniques. Le premier gros enjeu pour nous a donc été la réalisation d'une interface graphique : jusque-là nous n'avions eu l'occasion que de travailler sur des programmes s'exécutant en console. Le second reposait sur l'implémentation d'une IA fonctionnelle : elle doit pouvoir réfléchir intelligemment sur plusieurs coups, ce que nous n'avions jamais fait avant.

1.3.2 – Contraintes ergonomiques

La seule contrainte ergonomique pour ce projet fut tout simplement l'implémentation d'une partie graphique. Bien qu'aucune consigne ne nous ait été donnée, il fallait que le jeu soit facile à prendre en main directement par une personne n'y ayant jamais joué auparavant. Toutes les fonctionnalités devaient être claires et instinctives, c'est-à-dire compréhensibles sans avoir recours aux pages d'explications, toujours dans l'idée d'accrocher l'attention du joueur et de rendre agréable ses premiers instants de jeu.

2 – Rapport technique

2.1 – Conception

Etant libres pour le choix du langage de programmation, nous avons dressé la liste de plusieurs candidats possibles. De Java pour la portabilité au C^{++} que nous maîtrisions en passant par le Python pour la facilité et la rapidité, nous avons finalement retenu Ruby. Ce langage présente les avantages de C^{++} et Python tout en nous permettant d'apprendre un nouveau langage, que nous n'étudierons pas lors de notre cursus universitaire. Ce challenge nous a motivé à travailler encore plus que la normale afin d'être à niveau et ne pas prendre de retard.

Pour la bibliothèque graphique, nous avons suivi la même démarche : sélectionner plusieurs candidats possibles et n'en retenir qu'un. Celui-ci fut GTK. Une bibliothèque C^{++} également portée sur Ruby qui a l'avantage d'être prévue pour les petits comme les gros projets. Notre idée de départ étant de construire notre jeu étape par étape, cette bibliothèque peut donc accompagner notre projet au fur et à mesure qu'il prend de l'ampleur.

Nous avons également fait le choix unanime d'utiliser un gestionnaire de versions. D'une part, cela correspond encore une fois à notre choix de progresser par étape. D'autre part, à cause de la distance ou des emplois du temps différents, nous pouvions travailler sans la contrainte d'être réunis physiquement ni la crainte de ne pas être à jour.

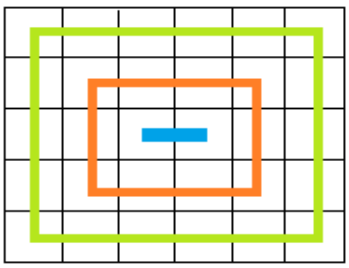
```ruby
def init_CellValue
#Initialisation de la ponderation de chacune des cellules, du bord vers le centre
  l1 = 0 # Indice du début de la ligne à pondérer
  l2 = @lines-1 # Indice de fin de la ligne à pondérer
  c1 = 0 # Indice de début de la colonne à pondérer
  c2 = @columns-1 # Indice de fin de la colonne à pondérer
  v = 1 # Valeur donnée a la case

  while (l1 <= @lines/2 || c1 <= @columns/2) do
  # Tant que l'on a pas parcouru tout le plateau
    for l in (l1..l2)
      for c in (c1..c2)
      # Pour chaque case
        if (l==l1 || l==l2) && (c==c1 || c==c2)
        # Si c'est un bord
          @board[l][c].value = v-1
          # Elle vaut un peu moins que les cases au même rang
        else
          @board[l][c].value = v
          # Sinon elle prend la valeur "classique"
        end
      end
    end
    # On parcourt le bord un cran plus petit
    l1+=1
    l2-=1
    c1+=1
    c2-=1
    v+=2 # La pondération des cases augmente lorsqu'on se rapproche du centre
  end
end
```

① ② ③

Figure 2 Algorithme de pondération des cases

① L'idée de cette fonction est de parcourir le plateau « bord après bord » :



On parcourt en premier les bords verts, puis les bords oranges et enfin les bords bleus.

Figure 3 Exemple de parcours "bord après bord"

Les premiers indices choisis sont donc ceux permettant de parcourir le bord du plateau (bord vert, figure 3).

② Pour chaque case, si elle représente un coin du bord et dans ce cas sa pondération est moindre qu'une autre case placée sur la même ligne ou colonne.

③ Ensuite, les indices sont modifiés afin de parcourir le bord plus petit (on passe du bord vert au bord orange, cf figure 3) et la pondération est augmentée comme on se rapproche du centre du plateau.

```
def logToGraph(x, y, image)
# Retrouve la cellule graphique associée à la bonne cellule logique et en change son image
  j = $game.board.columns - 1 # Indices des cases logiques
  i = $game.board.lines - 1

  for cell in $catable do # On itère sur chaque case graphique pour trouver celle où était placé le pion
    if(i == x && j == y) # Si les indices correspondent aux coordonnées (x, y) de la cellule cliquée)
      cell.picture.set(image) # On modifie l'image de la cellule (blanche ou occupée par un pion)
      break # Plus besoin de parcourir le reste du plateau
    end
    # Décrémentation manuelle simulant deux boucles for
    j = j - 1
    if j < 0
      j = $game.board.columns - 1
      i = i - 1
    end
  end
end
```

Figure 4 Correspondance entre les cases logiques et graphiques

Il arrive parfois qu'une case doit être modifiée sans que l'on ait l'information quant à sa position. Dans ce cas, une des solutions est de parcourir $catable, l'élément graphique qui contient et affiche des instances de la classe GraphicCell. La difficulté réside dans le parcours de $catable : celui-ci s'effectue de bas en haut et de droite à gauche, soit le sens inverse d'un parcours classique. Il a donc fallu recréer ce parcours avec deux indices représentant les coordonnées de la case recherchée pour pouvoir ainsi changer son image.

```
def iaCaseDepart
# Place l'ia sur une des cases les plus avantageuses du plateau
  lines = $game.board.lines # Facilite la lecture pour le reste de la fonction
  columns = $game.board.columns

  # On se place au milieu du plateau dans le sens de la longueur
  l1 = (lines%2 == 1) ? (lines/2+1) : (lines/2) # Indice de début des lignes
  l2 = lines/2+1 # Indice de fin des lignes
  # On se place au milieu du plateau dans le sens de la largeur
  c1 = (columns%2 == 1) ? (columns/2+2) : (columns/2+1) # Indice de début des colonnes
  c2 = columns/2 # Indice de fin des colonnes
  init = false # Vérifie si l'ia est placée
  alter = true # Permet d'affiner la recherche de la pondération la plus élevée

  while ((l1 > -1 || c1 > -1) && !init)do
  # Tant que l'on est dans le plateau et que l'ia n'est pas placée
    if (!alter)
    # On alterne l'incrémentation des indices afin de tester les coins en dernier
      l1-=1
      l2+=1
    else
      c1-=1
      c2+=1
    end
    alter = !alter

    for l in (l1..l2)
      for c in (c1..c2)
      # Pour chaque case
        if ($game.board[l1][c1].isAccessible) # Si elle est accessible
          init = true # L'ia se place dessus
        end
      end
    end
  end
  $game.board[l1][c1] # On renvoie la cellule choisie
end
```

①

Figure 5 Choix de la case de départ de l'IA

① Plus une case est située au centre du plateau, plus sa pondération est importante (cf figure 2) et donc il est préférable de s'y placer. C'est la raison pour laquelle, le parcourt du plateau commence par le centre.

Les indices l1 et c1 varient suivant si le nombre de lignes et de colonnes est pair ou impair. Le « bord » le plus au centre peut être un carré ou un rectangle (bord bleu cf figure 3)

```
# On initialise tous les elements necessaires a l'affichage et au commencement de la partie
def main

  gameInformations = Array.new(9) # On cree un tableau qui collectera toutes les informations necessaires a l'initialisation du jeu

  # On concerve on nombre de 2 joueurs
  gameInformations[0] = 2

  $endOfProgramm = true
# Booleen distinguant : _ lorsque l'on clique sur la croix pour fermant la fenetre pour mettre fin au programme -> 'true'
#                       _ lorque l'on ferme la fenetre en fin de partie pour en ouvrir une nouvelle (recommencer une partie) -> 'false'

  # Creation de la fenetre principale
  $window = Gtk::Window.new
  $window.set_title("ISOLA")         # Titre de la fenetre principale
  $window.set_default_size(350, 350) # Taille de la fenetre principal
  $window.window_position = Gtk::Window::POS_CENTER_ALWAYS # Quelque soit l'affichage, la fenetre sera toujours centree

  # Trouver comment mettre une image en fond pour le jeu
  background = Gtk::Image.new('images/background.jpg')

  # Boite verticale principale qui contiendra toutes les modifications d'affichage de la fenetre principale
  $allBox = Gtk::VBox.new

  $window.add($allBox) # On ajoute la boite principale a la fenetre principale
  $window.show_all     # Affichage de la fenetre principale et de ces 'enfants'

  # Lorsque la fenetre principale se ferme
  $window.signal_connect "destroy" do

    # Si l'on souhaite definitivement quitter le programme -> 'true'
    if $endOfProgramm

        Gtk.main quit # On quitte la boucle et le progamme s'arrete_
```

Figure 6 Initialisation des éléments graphiques

```
# Lorsque l'on clique sur le bouton 'Jouer' (cette operation sera recurrente, pour chaque bouton sur lequel l'utilisateur clique)
play.signal_connect "clicked" do

  # On supprime tous les elements(enfants/widgets) de la boite principale
  $allBox.remove(welcome)
  $allBox.remove(playAlignment)
  $allBox.remove(rulesAlignment)
  $allBox.remove(leaveAlignment)

  # On passe a l'etape(fonction) suivante en conservant notre tableau(en argument)
  # qui poursuivra sa collecte d'informations
  sizeOfBoard(gameInformations)

end # Fin des instructions liees au cliques du bouton 'Jouer'

# Lorsque l'on clique sur le bouton 'Regles'
rules.signal_connect "clicked" do

  $allBox.remove(welcome)
  $allBox.remove(playAlignment)
  $allBox.remove(rulesAlignment)
  $allBox.remove(leaveAlignment)

  displayRules(gameInformations)

end # Fin clique 'Regles'

# Lorsque l'on clique sur le bouton 'Quitter'
leave.signal_connect "clicked" do

  $allBox.remove(welcome)
  $allBox.remove(playAlignment)
  $allBox.remove(rulesAlignment)
  $allBox.remove(leaveAlignment)
```

Figure 7 Action des boutons du menu

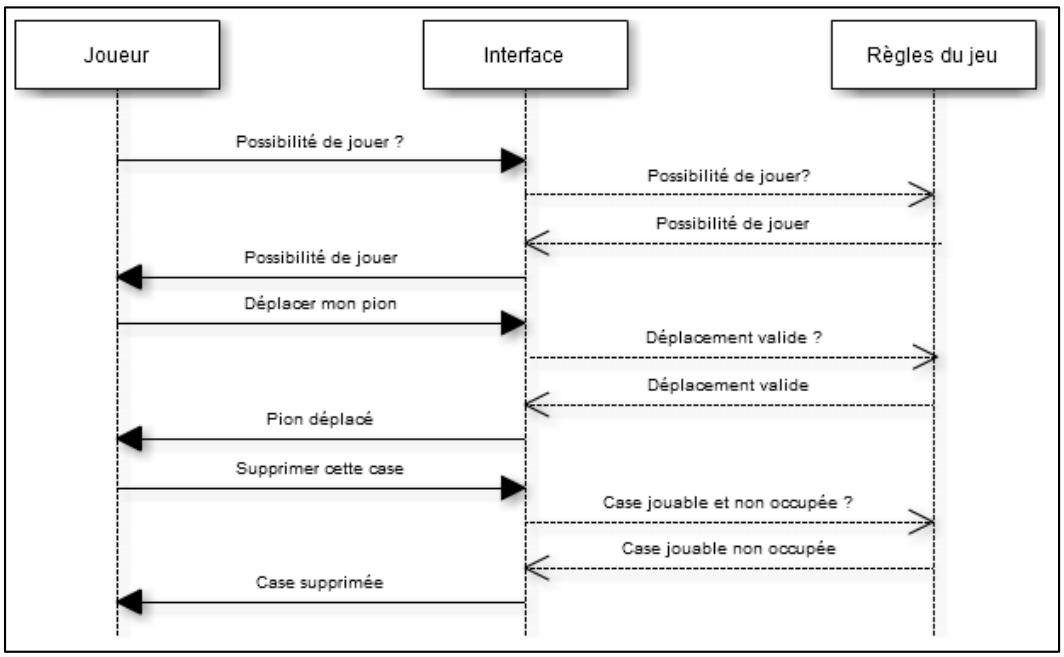


Figure 8 Diagramme de séquence du tour basique d'un joueur

Pour avoir une idée des principales fonctions à implémenter, nous avons réalisé un diagramme de séquence qui détaille comment le programme et le joueur interagissent entre eux. Nous avons donc noté trois étapes fondamentales dans le tour d'un joueur : sa possibilité de jouer, le déplacement de son pion et la suppression d'une case jouable. Tout cela en respectant évidemment les règles du jeu.

2.2 – Résultat

Notre objectif est donc de programmer le jeu Isola selon les règles données, pour deux joueurs ou un joueur et l'ordinateur.

Notre projet, dans sa version présentée pour l'examen, remplit ces conditions.

Nous avons pu par la suite rajouter une fonctionnalité : les bonus.

2.3 – Perspectives de développement

Avant de commencer à attaquer la partie technique du projet, nous avons beaucoup réfléchi à la direction à prendre. Que ce soit pour les outils utilisés ou

l'implémentation de nouvelles fonctionnalités, les idées ont été nombreuses. Une liste d'améliorations avait été rédigée :

- La modification de la taille du plateau ainsi que sa forme (éditeur de niveau) ;
- L'amélioration de l'interface des règles du jeu (rendu dynamique) ;
- La possibilité de sauvegarder une partie en cours ;
- La possibilité de jouer à plus de deux joueurs ;
- Le rajout de bonus lorsqu'un pion se place sur une case ;
- L'implémentation d'un mode histoire (joueur contre des IA de plus en plus compétentes) ;
- L'adaptation du jeu pour les terminaux Android.

Avec plus de temps, nous aurions aimé ajouter ces fonctionnalités qui permettent une grande rejouabilité pour le jeu.

4 – Rapport d'activité

4.1 – Cycle de développement

Pour notre organisation, nous avons décidé de progresser par étapes et ainsi de réaliser plusieurs versions du jeu, chacune ayant une fonctionnalité supplémentaire par rapport à l'ancienne.

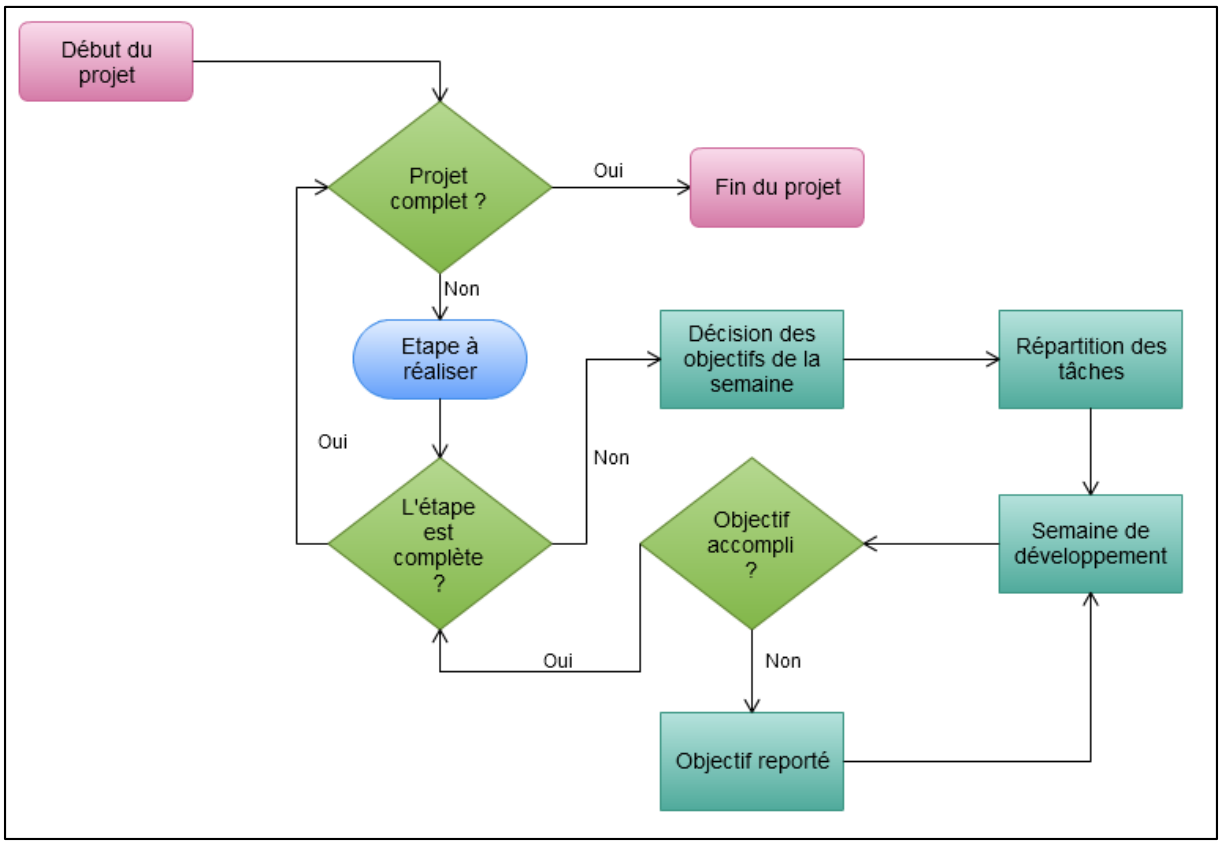


Figure 9 Cycle de développement

Notre démarche était simpliste. Premièrement, nous nous sommes fixés une première étape à réaliser : une version joueur contre joueur. Les objectifs ont d'abord été de l'ordre de la réflexion : diagramme des classes, brouillon d'algorithmes, discussion sur les méthodes à utiliser ou encore étude de la bibliothèque graphique. Ensuite sont venus les objectifs pratiques : implémentation des classes et de l'interface.

Une fois cette version stable, les fonctionnalités suivantes se sont ajoutées : la possibilité de jouer contre l'IA, la finition du menu ainsi qu'une étape que nous avions imaginé : les bonus.

Afin d'avoir toujours à disposition notre travail, nous avons décidé d'adopter un gestionnaire de versions. Nous avons donc utilisé GitHub tout au long de notre projet, ce qui nous a permis de travailler séparément une fois les tâches réparties ou tout simplement de pallier le problème de la distance.

4.2 – Planification

Lors de la première réunion avec notre tuteur, il nous a fortement recommandé d'établir un planning prévisionnel ayant pour but de nous faire une idée sur le travail à fournir et le temps qu'il faudrait pour le réaliser.

Figure 10 Planning prévisionnel et réel du projet

Nous avons donc mis en place le planning suivant :

- Trois semaines pour le travail en amont : réfléchir au projet, rédiger les diagrammes UML, faire la liste des développements possibles, choisir nos outils et se familiariser avec eux ;
- Un mois pour la première version : la rédaction des classes, la prise en main de la bibliothèque graphique, la création d'un main, les phases de tests ainsi que le lien entre la partie graphique et la partie logique ;
- Une semaine pour la seconde version : le plus gros du travail a été réalisé lors de la phase précédente. Ici la principale difficulté résidait dans les phases de débogage.
- Le reste du temps disponible pour les fonctionnalités optionnelles : une fois notre contrat rempli, tout le temps restant devait être utilisé pour implémenter les règles que nous avions inventées.

Comme cela est visible, réaliser la première version nous a pris plus de temps que prévu. Ce retard est dû à notre volonté de vouloir travailler ensemble bien que nous avions décidé de nous répartir les tâches. En effet, même si certaines fonctions sont indépendantes, nous ne voulions pas prendre le risque que certaines personnes soient en retard par rapport à d'autres. Il a donc fallu trouver le temps de nous réunir ou au moins de pouvoir discuter posément de nos avancées sur les différentes parties du jeu.

Conclusion

Le cahier des charges comprenait les objectifs suivant :

- Réaliser une version du jeu Isola jouable par deux personnes ;
- Réaliser une version du jeu Isola jouable par une personne contre l'ordinateur ;
- Réaliser une interface graphique du jeu Isola.

Ces objectifs ont tous été accomplis : les deux modes de jeux sont disponibles et le rendu graphique est satisfaisant. Nous avons, en plus, rajouté une fonctionnalité qui permet aux joueurs de récupérer des bonus s'ils se placent sur certaines cases.

Les possibilités de développement sont diverses. Nous avions dressé une liste de ce que nous pouvions implémenter lors de notre phase de réflexion. Elle résulte surtout en l'amélioration graphique du jeu, l'ajout de règle ou le portage sur terminaux Android.

Ce projet a été pour nous une grande occasion d'apprendre de nouvelles techniques. La gestion d'une bibliothèque graphique étant tout nouveau pour nous il a fallu se renseigner durant de nombreuses heures, à cause du manque de documentation pour GTK Ruby. Apprendre un nouveau langage était également un challenge intéressant. Cela n'a au final pas représenté l'obstacle que nous craignions et ce fut au final un plaisir.

Pour ce qui est de l'apprentissage des méthodes de travail, cela n'a pas été concluant à cause de notre manque d'expérience.